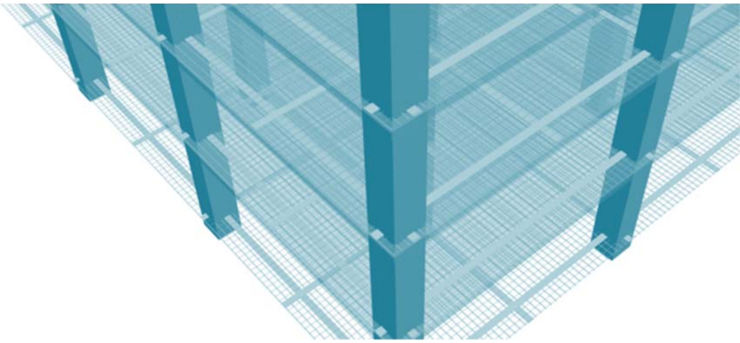
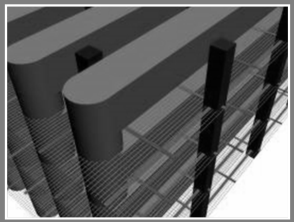
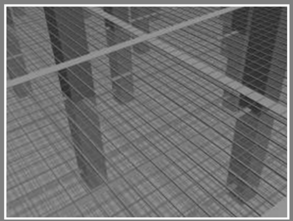
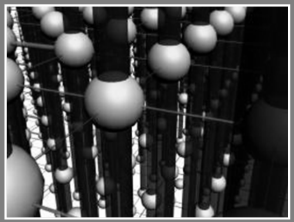


Flow Based Min-Cut Balanced Bipartitioning Implementation

Bon Woong Ku

Kyungwook Chang

Kartik Acharya

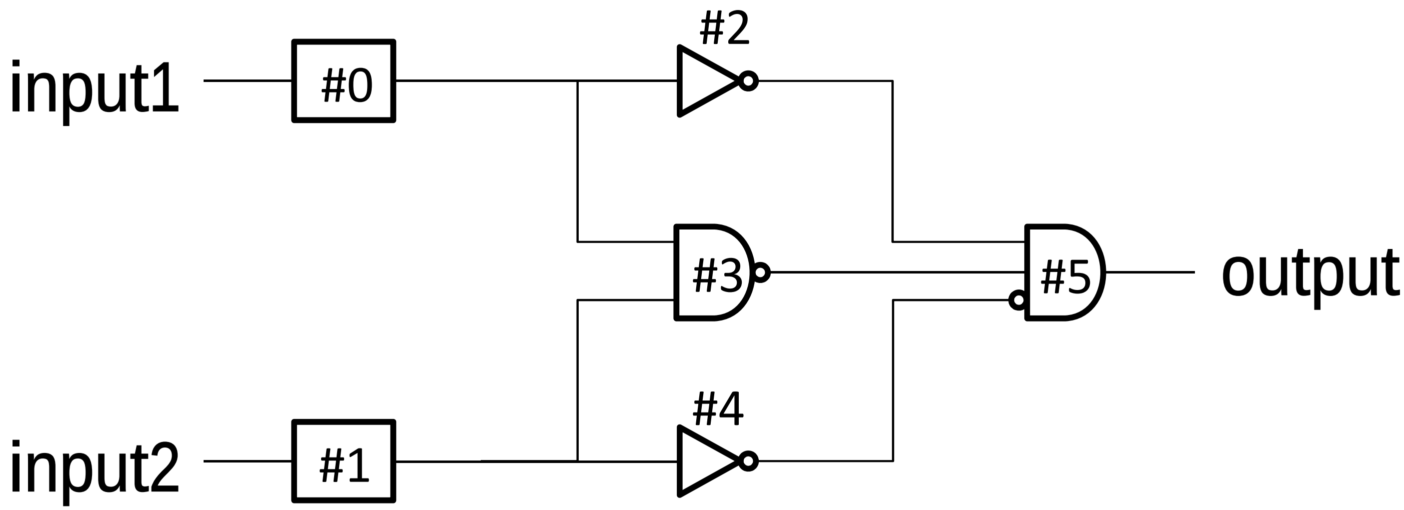
Contents

- Algorithm steps with a small circuit
  - Max-flow computation engine = Dinic's blocking flow algorithm
- How to handle unreachable nodes
- Our implementation features
  - Speed and accuracy
- Partitioning, and skew impact result
- Possible extensions

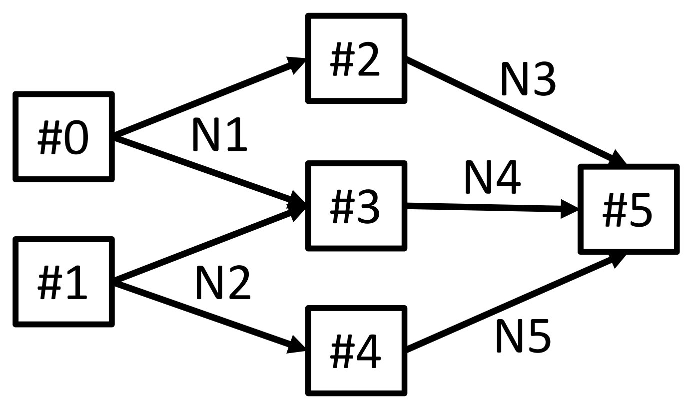
1. Parsing and extracting netlist from .blif

- Cell's weight = 1, Skew = 0%, ratio = 0.5

```
.model simple
.inputs i0 i1
.outputs o0
.latch i0 a 0
.latch i1 b 0
.names a c
0 1
.names a b d
1 1 0
.names b e
0 1
.names c d e o0
1 1 0 1
.end
```

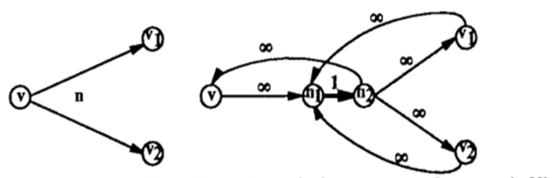


- Netlist extraction
  - N1(0:2,3), N2(1:3,4), N3(2:5), N4(3:5), N5(4:5)

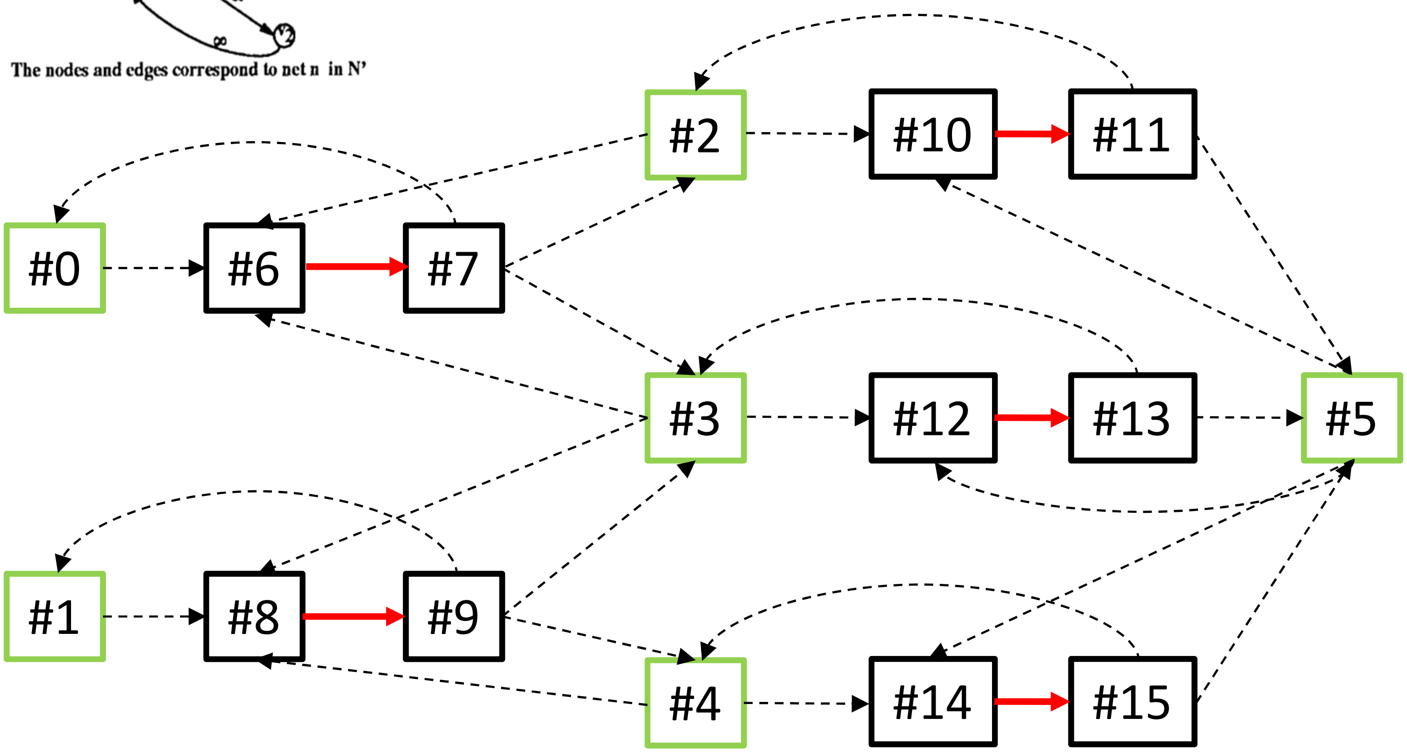


2. Modeling the netlist to the flow network

- → means unit capacity, ⇢ means infinite capacity
- □ (green) means actual cell nodes, □ (black) means added nodes

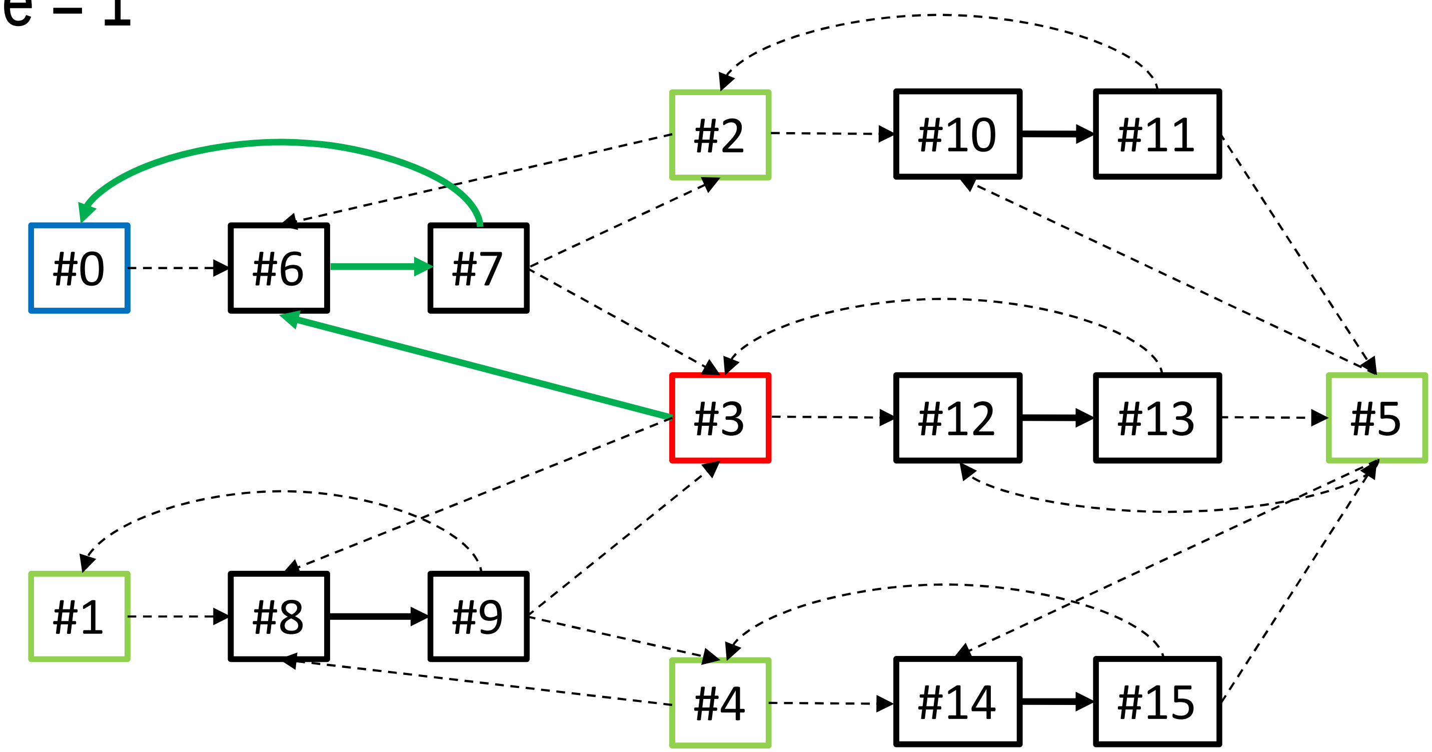


A net n in circuit N The nodes and edges correspond to net n in N'



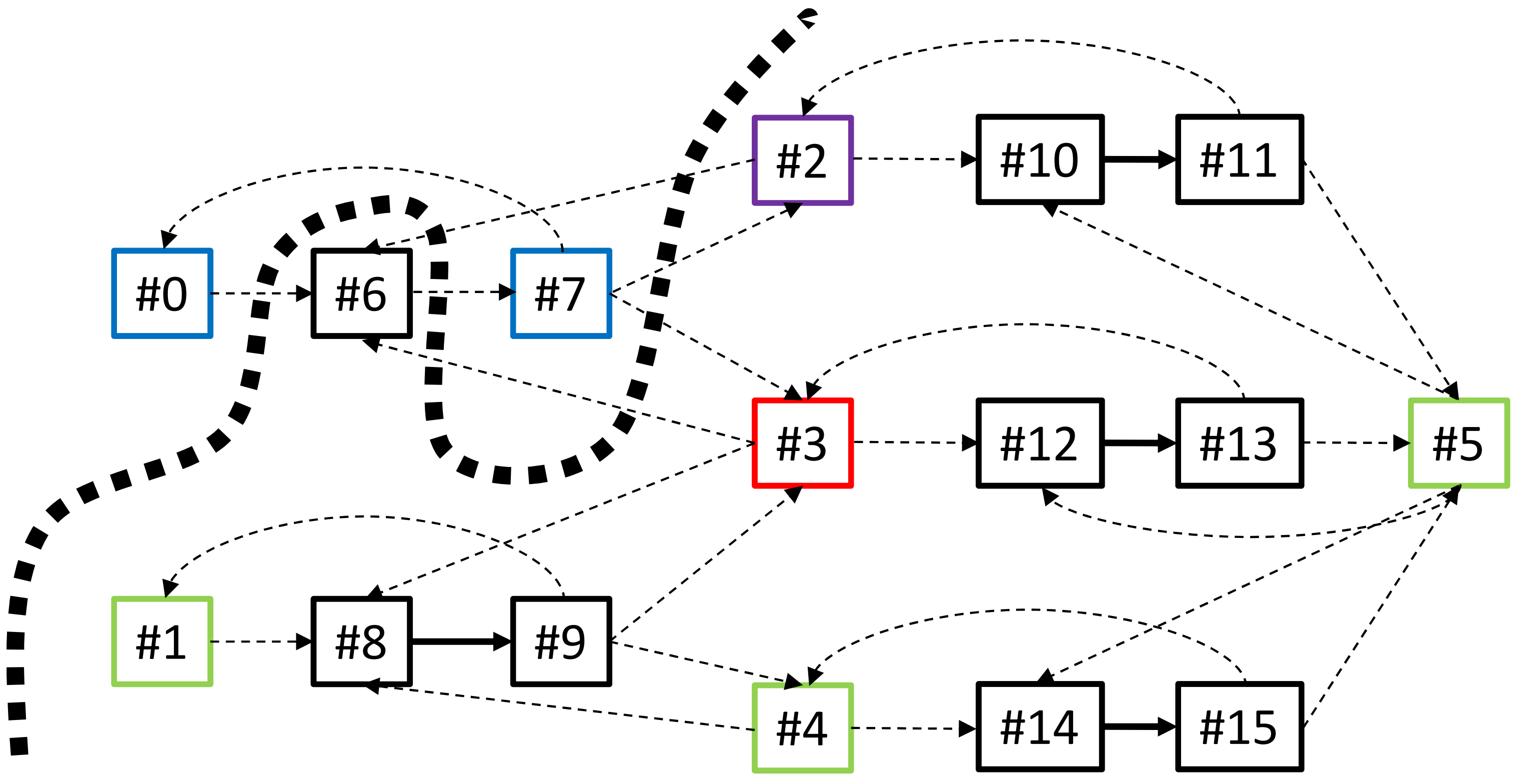
3. Pick (S,T) pair, and max-flow computation

- Randomly pick a pair of cell node (S,T)
  - S = #3, T = #0
- Find maximum flow network
- Cutsize = 1



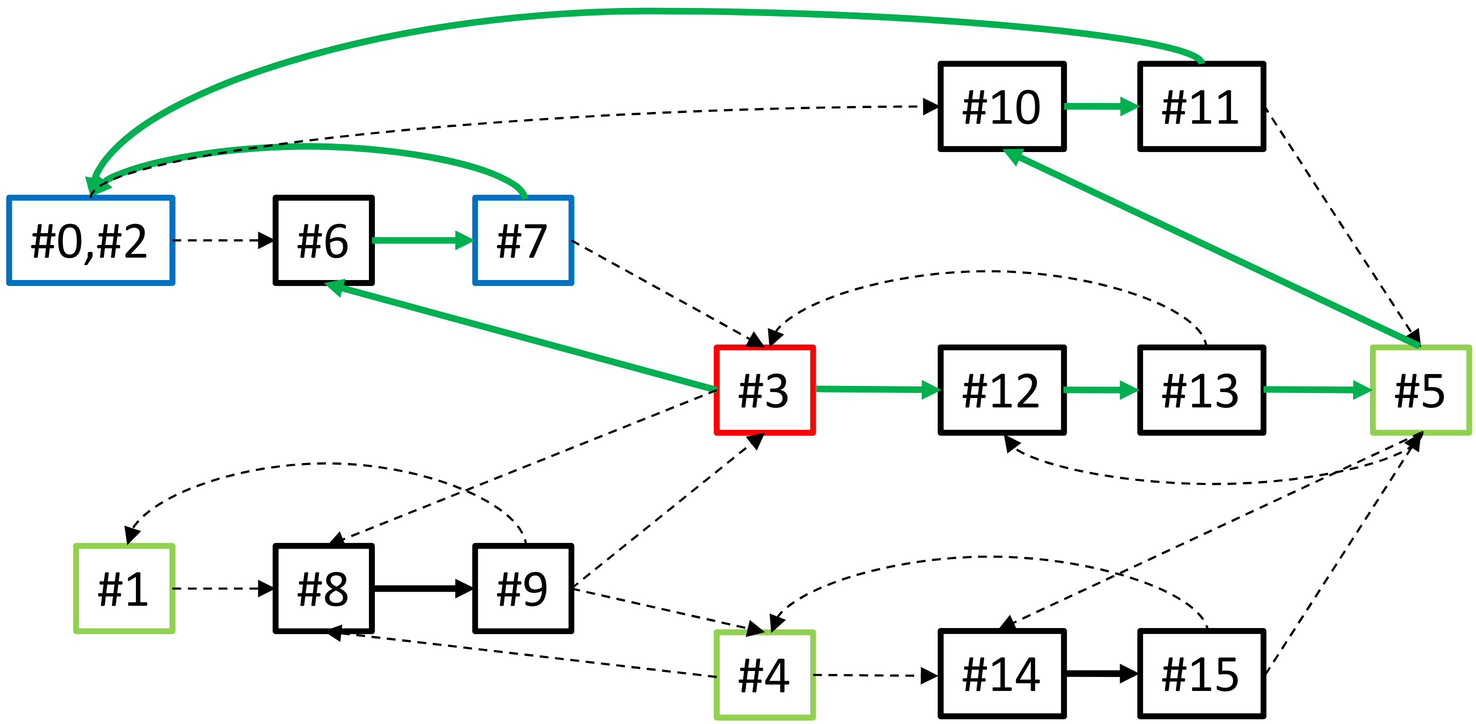
4. s-t cut(X,X') Bipartition and node collapsing

- X = {1,2,3,4,5,6,8,9,10,11,12,13,14,15} and X' = {0,7}: unbalanced!
- Collapse all nodes in X' to T
- Net(X,X') = N1(0:2,3), adjacent node = #2



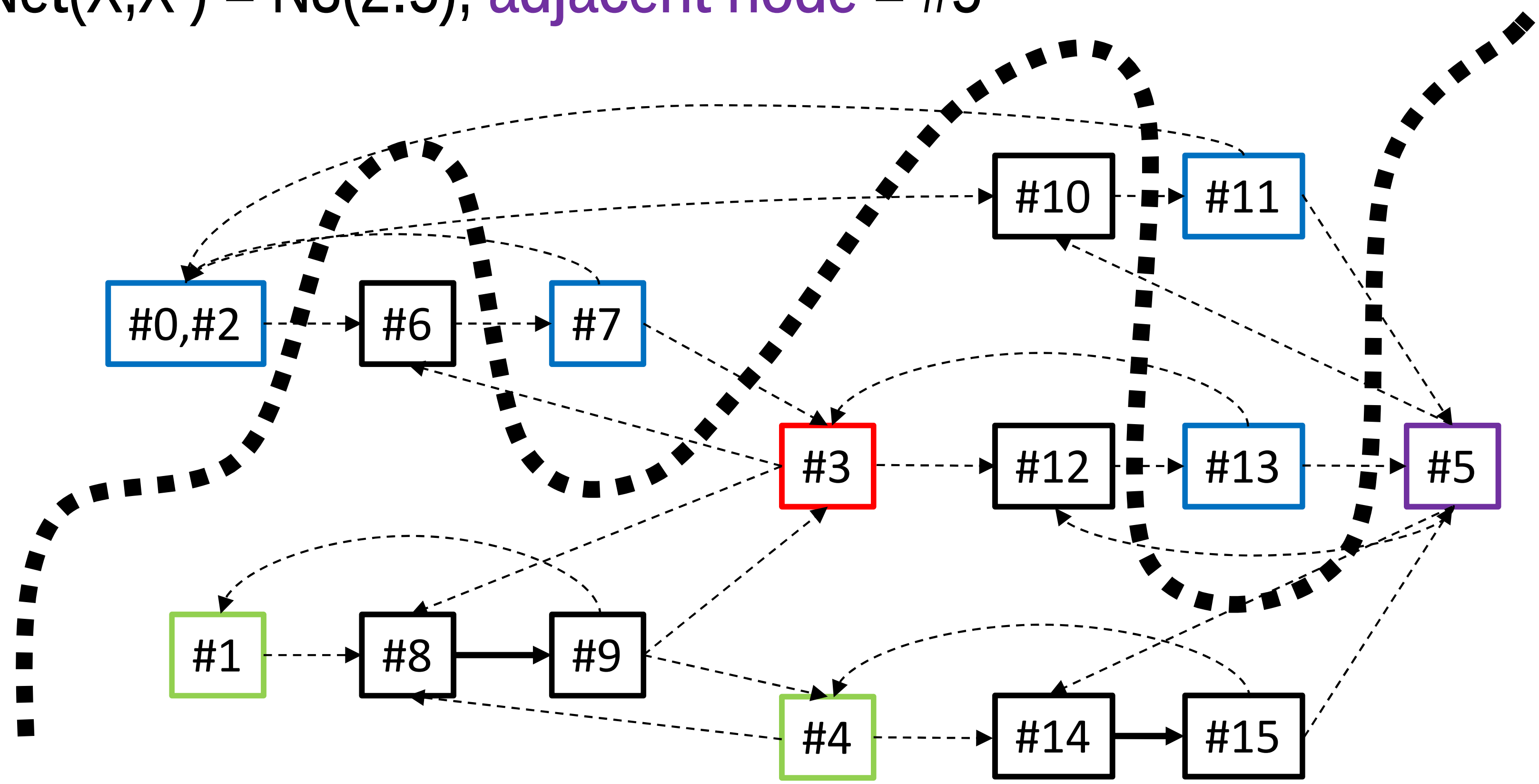
5. Max-flow computation

- After collapse #2 to X’
  - S = #3, T = #0,#2,#7
- Find maximum flow network
- Cutsize = 2



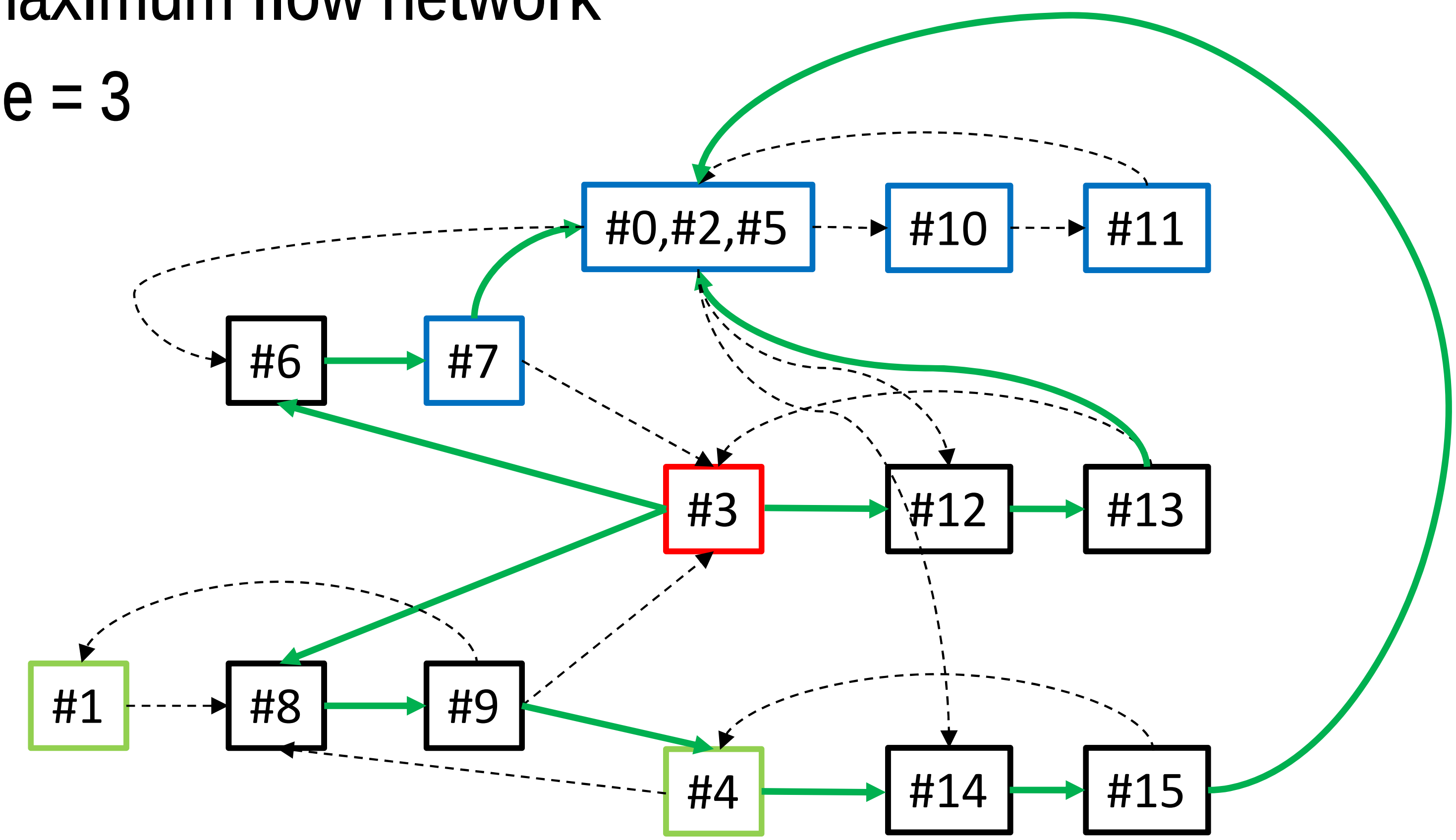
6. Bipartition and collapsing nodes

- X = {1,3,4,5,6,8,9,10,12,14,15} and X' = {0,2,7,11,13}: unbalanced!
- Collapse all nodes in X' to T
- Net(X,X') = N3(2:5), adjacent node = #5



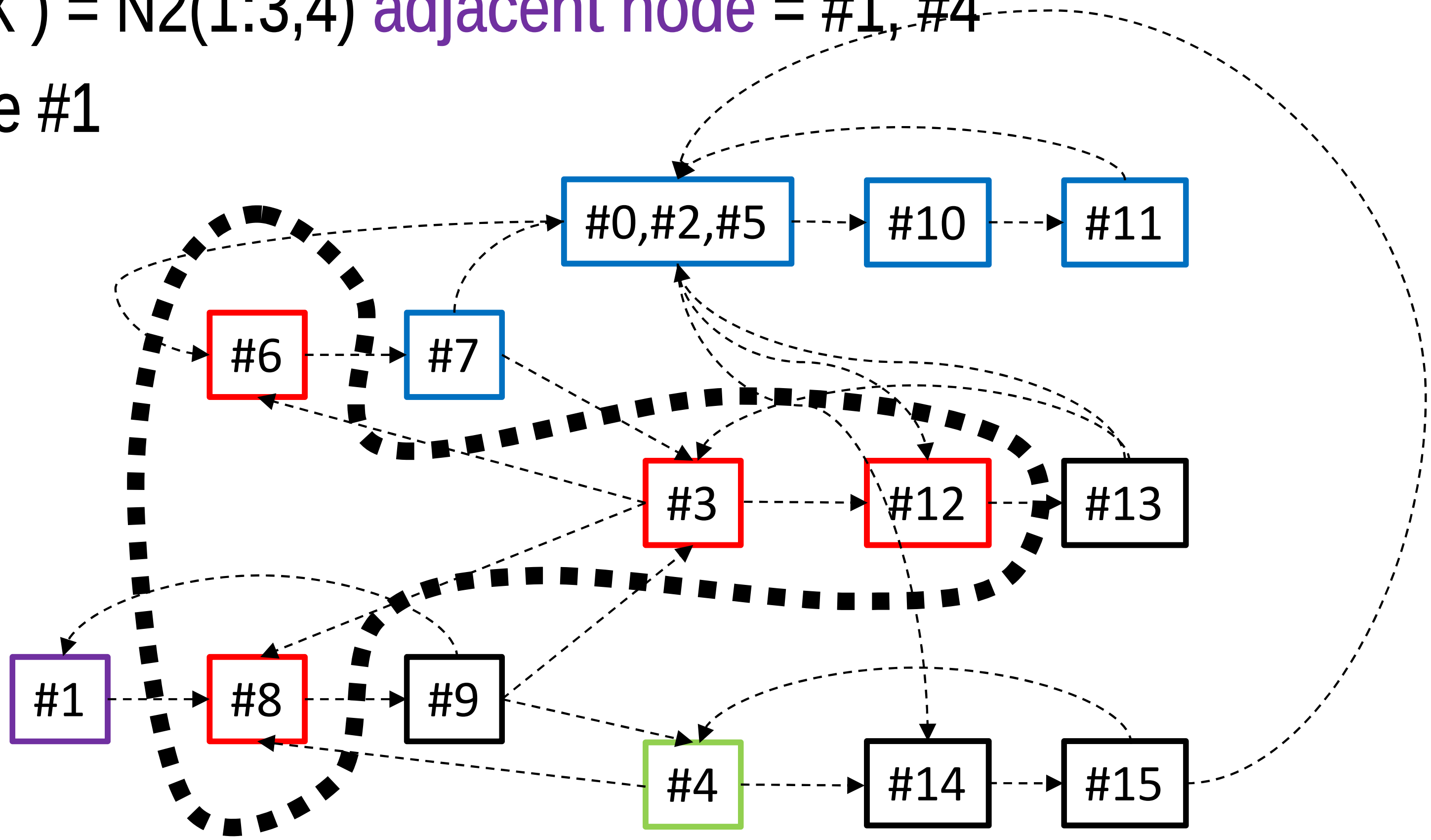
7. Max-flow computation

- After collapse #5 to X’
  - S = #3, T = #0,#2,#5,#7,#10,#11
- Find maximum flow network
- Cutsize = 3



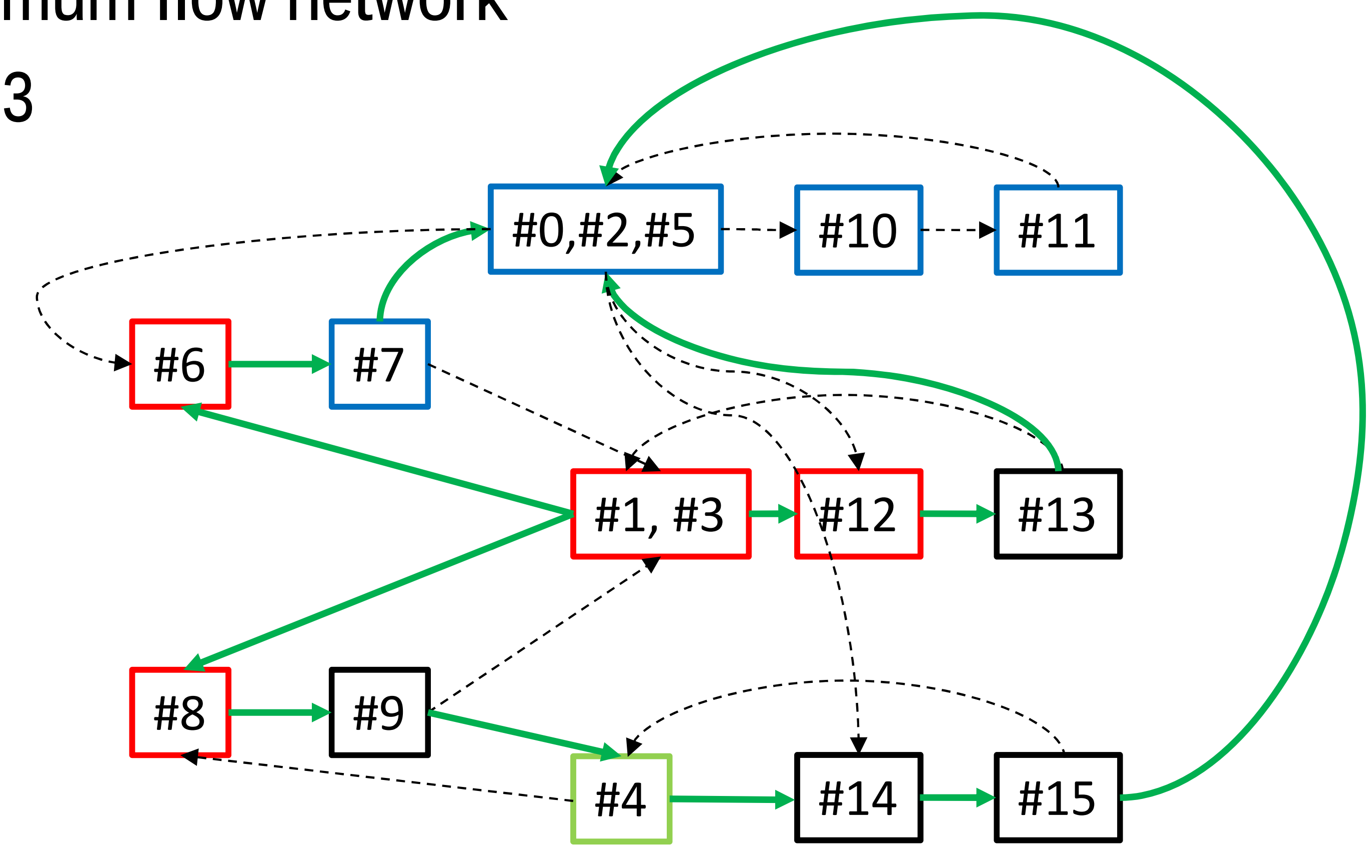
8. Bipartition and collapsing nodes

- X = {3,6,8,12} and X’ = {0,1,2,4,5,7,9,10,11,13,14,15}: unbalanced!
- Collapse all nodes in X to S
- Net(X,X’) = N2(1:3,4) adjacent node = #1, #4
- Choose #1



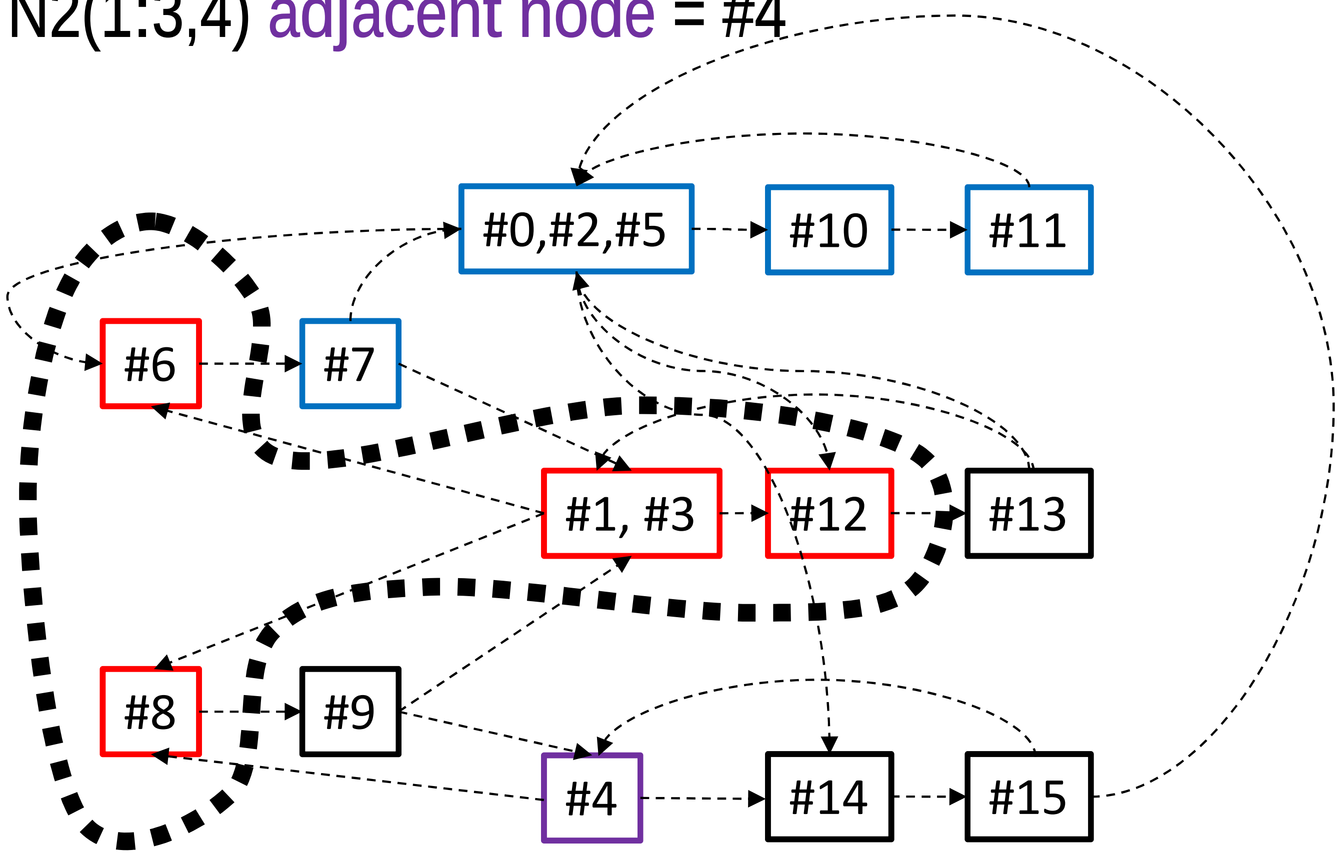
9. Max-flow computation

- After collapse #1 to X
  - S = #1,#3,#6,#8,#12 T = #0,#2,#5,#7,#10,#11
- Find maximum flow network
- Cutsize = 3



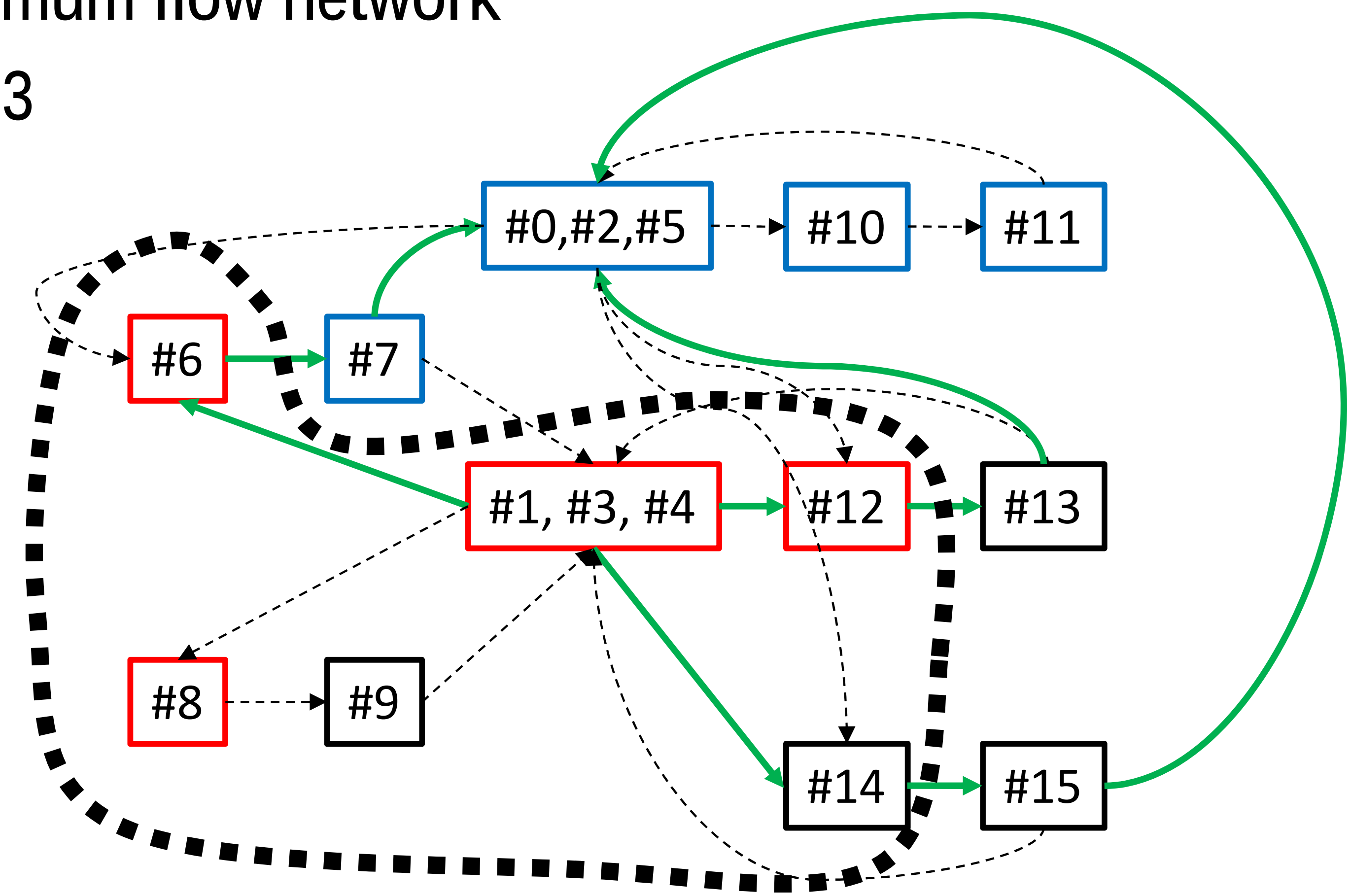
10. Bipartition and collapsing nodes

- X = {1,3,6,8,12} and X' = {0,2,4,5,7,9,10,11,13,14,15}: unbalanced!
- Collapse all nodes in X to S
- Net(X,X') = N2(1:3,4) adjacent node = #4

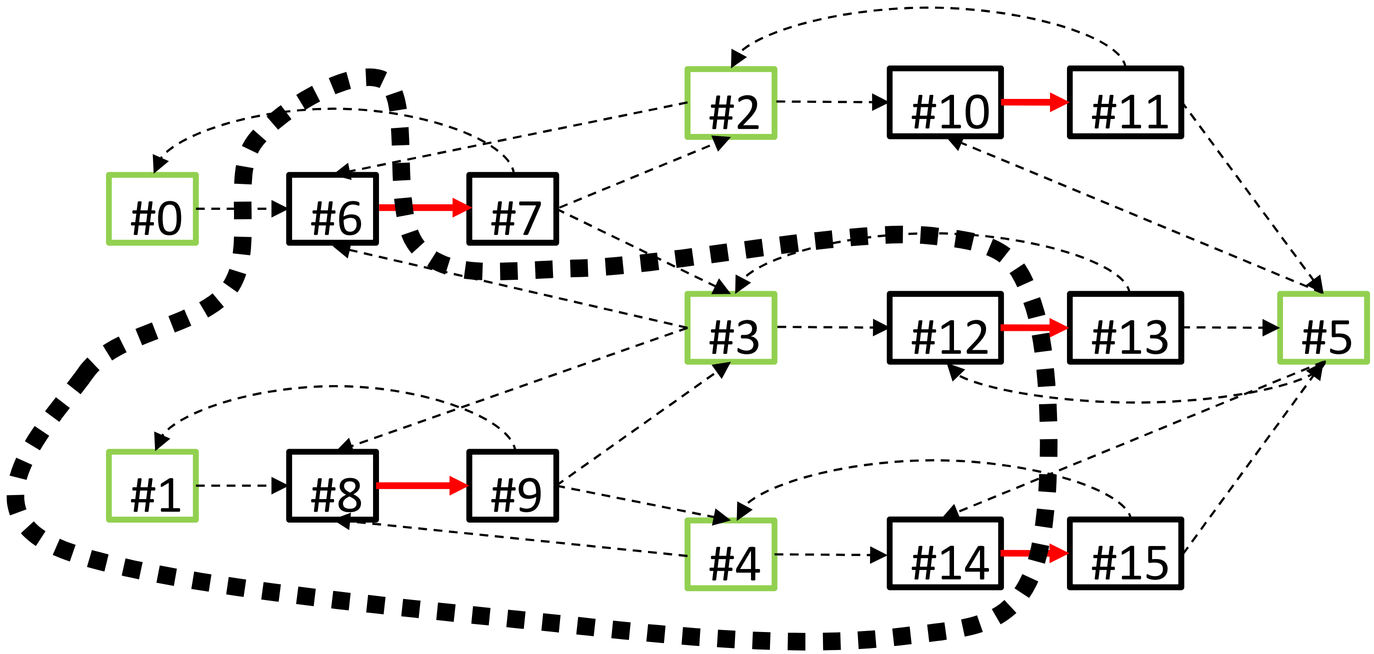


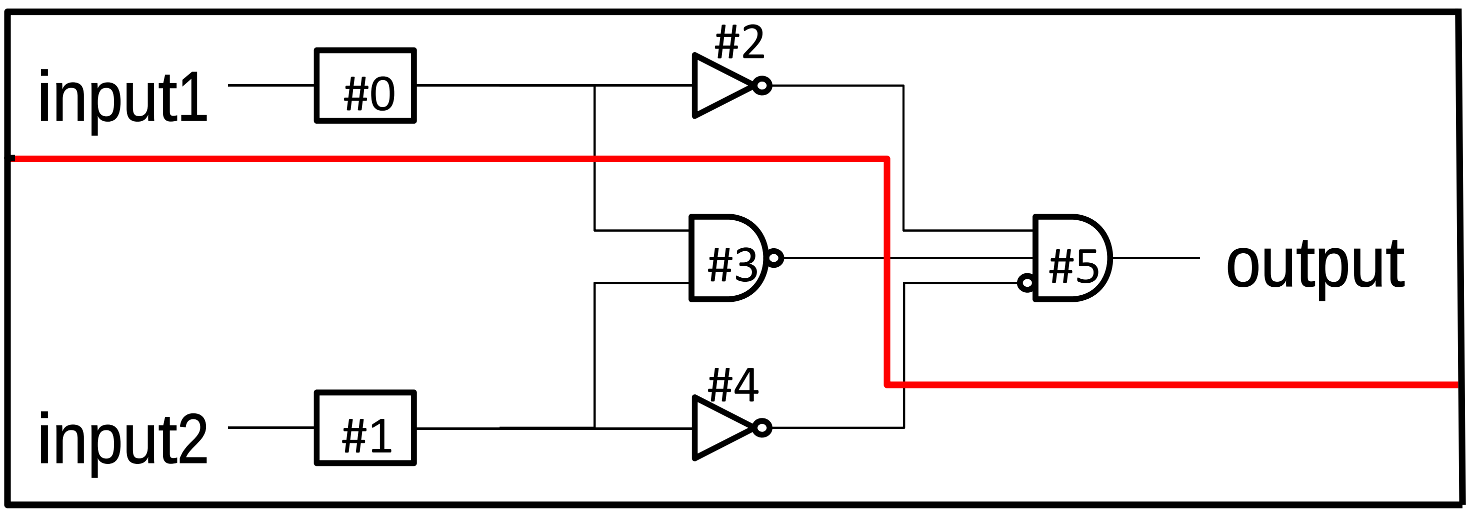
11. Max-flow computation

- After collapse #4 to X
  - S = #1,#3,#4,#6,#8,#12 T = #0,#2,#5,#7,#10,#11
- Find maximum flow network
- Cutsize = 3



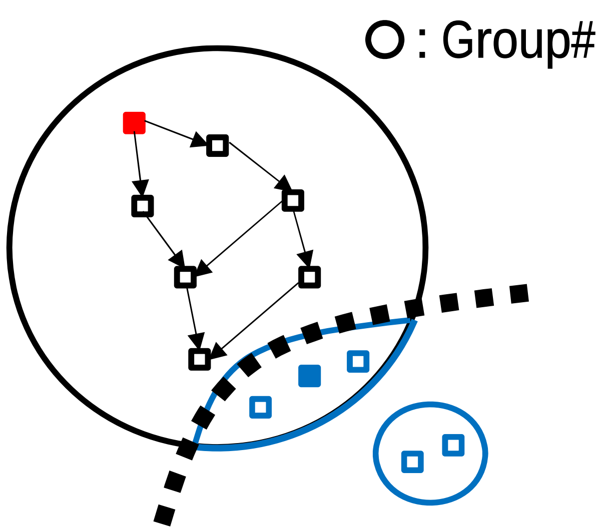
12. Final balanced bipartition





Handling unreachable nodes

- There were unreachable node groups in s13207, s9234 circuits
  - We checked this by running DFS and BFS.
  - Result: (G# = group#)
    - s13207: G#1 = 8486, G#2 = 9, G#3 = 71, G#4 = 13, G#5 = 28, G#6=107
    - s9234: G#1 = 5787, G#2 = 9, G#3 = 22, G#4 = 13, G#5 = 13
- However, it doesn't matter in FBB algorithm once we choose the S-T pair in the largest group



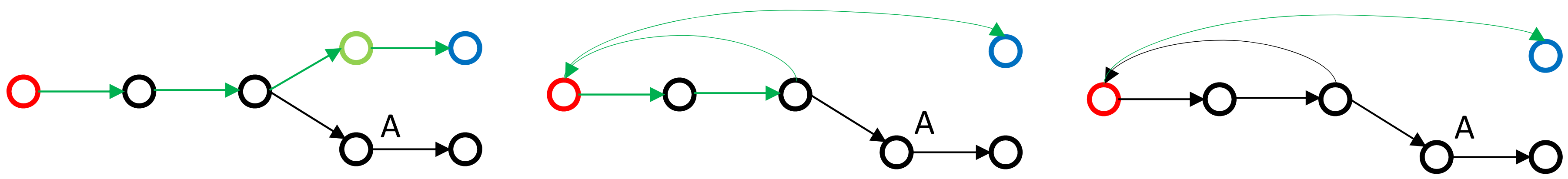
1. X is the set of reachable nodes from S through augmenting path.
2. If there are unreachable nodes in the network, it is always considered as X'.
3. Since small partition is collapsed into one nodes when partition is not balanced, unreachable nodes also would be collapsed into small partition

Our implementation feature

- To make it fast
  - We used Dinic's blocking flow algorithm for max-flow computation
    - It is suitable for sparse flow graph. A cell has limited number of nets
    - Although time complexity is $O(n^2 \log n)$, it's quite fast in practice
  - We handled connectivity, capacity and flow of a graph using STL vectors, Dinic's algorithm, and collapsing node runs directly on the vectors
    - Using matrix gives us $O(n^3)$
    - Time complexity is now $O(n^2 \log n)$
      - B17 bechmart partitioning runtime comparision
        - » Using matrix: 42260s, Our method: 4307s
  - We picked up the adjacent node to be collapsed faster
    - We didn't checked the all possible adjacent node
    - Instead, we checked the first reachable adjacent node

Our implementation feature

- To make it accurate
  - Cycle removal
    - When we collapse the nodes, it could be possible to make invalid flow graph



    - With cycle removal, we can reach to node A now
  - We completed basic FBB implementation
    - We can give options including skew and balancing ratio to our program

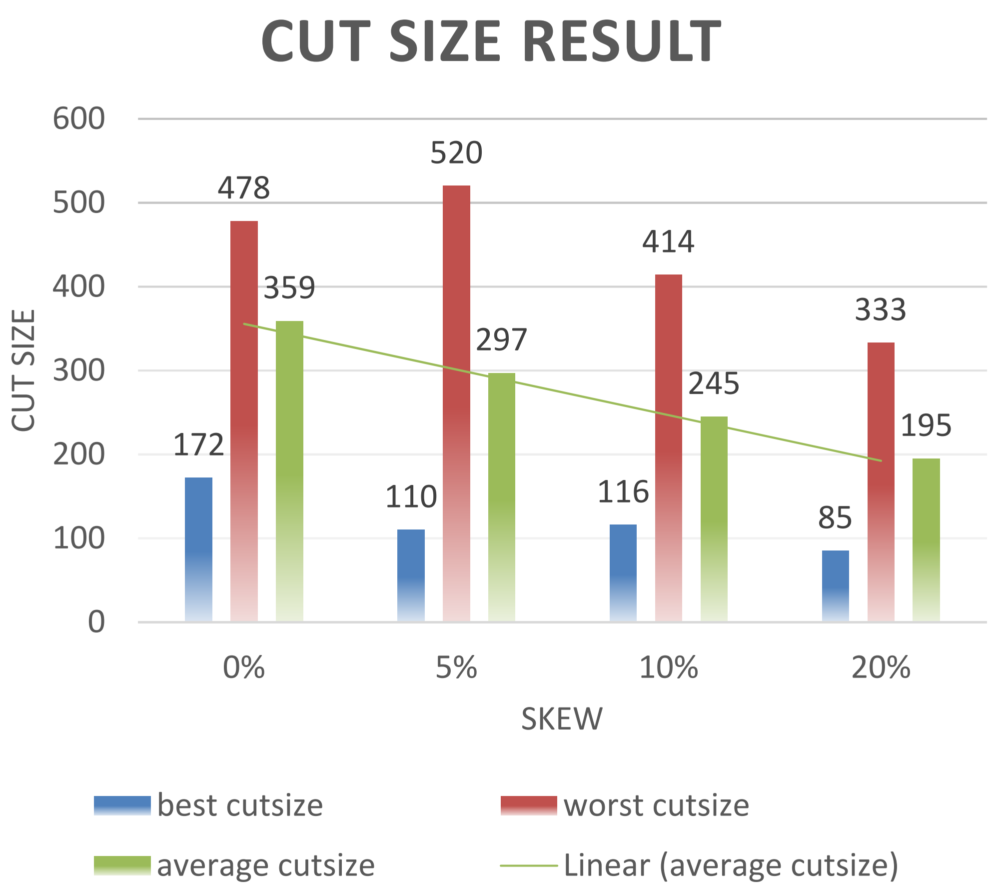
Best partitioning result

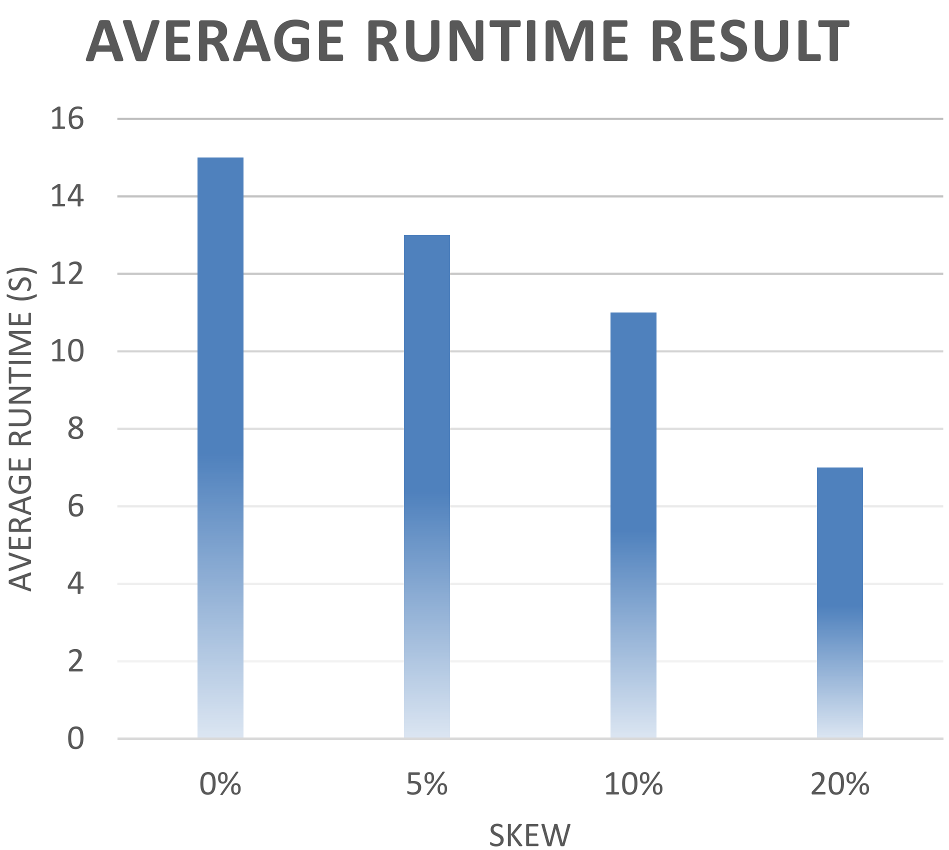
- Ratio = 0.5, skew = 5%

	S13207	S9234	b20_opt	b22_opt	b17_opt
Input #	31	36	32	32	37
Output #	121	39	32	22	46
Cell #	8696	5808	11979	17351	22854
Best cut size	98	160	365	980	811
Partition 1	4185	2759	6208	8683	11539
Partition 2	4511	3049	5771	8668	11315
Execution time(s)	4	9	5	100	140
Worst cut size	375	530	2260	3662	3678
Partition 1	4131	3020	5720	8244	11452
Partition 2	4565	2788	6259	9107	11402
Execution time(s)	98	19	122	327	474
Average cut size	204	271	1390	2047	1726
Execution time(s)	19	11	91	192	228
Cell# / (cut size X time)	2.24	1.95	0.09	0.04	0.06

Skew impact result

- Benchmark: S9234, #cell = 5808





Expecting extensions

- Compare with KL, FM partitioning result
- Show the impact of s-t selection on cutsize
  - Random s-t selection
  - Max-min-path s-t selection using modified Dijkstra's algorithm
    - Modified Dijkstra's algorithm finds min-path to all other nodes from source, with same time complexity of Dijkstra's algorithm
    - We choose a sink which has maximum min-path from source